Umgang mit Datensätzen

Daten bändigen & visualisieren mit 

B. Philipp Kleer
Methodentage 2021
11. Oktober 2021

Tidyr und dplyr

Daten können in verschiedenen Formaten vorliegen. Die zwei bekanntesten sind wohl das

1. *long*-Format
2. *wide*-Format (*standard*)

Aber auch andere Formate für Daten sind möglich. Z.B. wenn wir Daten über eine API laden. Auch bei Zusammenführen von Datensätzen muss das Format beachtet werden.

Diese verschiedenen Varianten werden wir uns nun in `tidyverse` mithilfe von `tidyr` und `dplyr` anschauen.

`

tidyr und Datensatz-Formate

Das Gerüst von Datensätzen mit tidyr

Es ist für die Verarbeitung in R wichtig, dass die Datensätze *tidy* sind, damit die Funktionen in R problemlos mit den Daten laufen. Zum Beispiel für die Anwendung von `ggplot` empfehlen sich *tidy*-Datensätze. Das bedeutet, dass die Daten in einem bestimmten Format vorliegen müssen, damit die Funktionen in R auch gut mit den Daten funktionieren (weniger *troubleshooting*).

Was ist ein Datensatz?

Ein Datensatz ist generell immer eine Sammlung von Werten, sei es numerisch oder eine Zeichenkette. Diese Werte sind immer auf zwei Arten organisiert: Jeder Wert gehört zu einer **Variable** und zu einer **Beobachtung**. Eine **Variable** inkludiert alle Werte, die für diese gemessen worden sind (also alle Beobachtungen auf dieser Variable). Eine **Beobachtung** inkludiert alle Werte, die für diese Beobachtung gemessen wurden (also alle Variablenwerte dieser *Einheit*).

Damit Daten in R gut mit den Funktionen genutzt werden können, müssen diese in einem *tidy*-Format vorliegen (auch *long*-Format genannt). Ein Datensatz ist dann *tidy*, wenn ...

- ... jede Variable eine Spalte ist,
- ... jede Beobachtung eine Zeile ist,
- ... und jede Beobachtungseinheit eine Tabelle formt.

untidy data sets (Beispiel 1)

Im Folgenden bearbeiten wir zwei Datensätze, die jeweils nicht *tidy* sind.

```
statclass
```

```
    name stat1 stat2  r spss
1   momo    12     5  6    9
2    kim    14    10 13   15
3 sascha     7     4  4    1
```

Dieser Datensatz ist im sogenannten *wide*-Format. D.h. wenn wir neue Prüfungen hätten, würden wir einfach weitere Spalten hinzufügen. Dies ist aber für die Verarbeitung mit R teilweise problematisch, denn wir benötigen oft ein *long*-Format.

`

untidy data sets (Beispiel 2)

```
statclass2
```

	test	momo	kim	sascha	exam
1	stat1	12	13	4	exam1
2	stat1	NA	NA	8	exam2
3	stat2	5	10	5	exam1
4	stat2	NA	NA	NA	exam2
5	r	6	13	3	exam1
6	r	NA	NA	9	exam2
7	spss	9	4	7	exam1
8	spss	NA	7	NA	exam2

In diesem Fall haben wir mehrere Probleme: Zum einen sind in den Spalten nicht überall Variablen, sondern Beobachtungen (`momo`, `kim`, `sascha`) und in `exam` finden wir wiederum Variablennamen.

Fangen wir mit `statclass` an.

`

Tidy-up statclass

In der Tabelle kann die Note jeder Person aus jeder Prüfung ausgelesen werden. Überlegt kurz, welche Variablen wir bei diesem Satz generieren möchten!

```
statclass
```

```
    name stat1 stat2  r spss
1   momo    12     5  6    9
2    kim    14    10 13   15
3 sascha     7     4  4    1
```

- `names`: momo, sascha, kim
- `course`: statI, statII, r, spss
- `grade`: Wert in Abhängigkeit der zwei oberen.

Es sind also zwei Informationen in den Spalten `stat1`, `stat2`, `r` und `spss`. Nämlich welcher Test es ist (implizit über Variablenname) und die Note. D.h. hier sind Werte als Variablenname angegeben und das verstößt gegen die Regeln eines *tidy* Datensatzes. Wir benötigen in einem *tidy*-Format aber beide Informationen explizit! Denn die Spaltennamen sind hier Werte (Art der Prüfung) und nicht einfach Namen.

`

Tidy-up statclass

Um dies zu bereinigen, nutzt man `pivot_longer()`. Hierbei geben wir zuerst an, welche Spalten neugeordnet werden sollen (in unserem Fall `stat1` bis `spss`), dann in welche neuen Variablen die Namen bzw. die Werte gespeichert werden sollen. Mit `names_to` benennen wir die neue Variable, die den Test unterscheidet und mit `values_to` benennen wir die Variable, die die Noten beinhaltet.

```
statclassTidy <- statclass %>%
  pivot_longer(stat1:spss,
               names_to = "course",
               values_to = "grade"
               ) %>%
  arrange(name,
          course
          )

statclassTidy
```

```
# A tibble: 12 × 3
   name   course grade
   <chr>  <chr>  <dbl>
 1 kim    r         13
 2 kim    spss      15
 3 kim    stat1     14
 4 kim    stat2     10
 5 momo   r          6
 6 momo   spss       9
 7 momo   stat1     12
 8 momo   stat2      5
 9 sascha r          4
10 sascha spss       1
11 sascha stat1      7
12 sascha stat2      4
```

Jetzt haben wir ein *long*-Format, dass die Datenbearbeitung oft einfacher macht (z.B. mit `ggplot2`). **Aber Aufpassen**: Man kann jetzt nicht einfach mehr einen Mittelwert von `grade` berechnen, da dies verschiedene Kurse beinhaltet. Man muss dabei also Bedingungen setzen (wenn man im *long*-Format ist).

Back to wide

Möchte man dies wieder umkehren, nutzt man die Funktion `pivot_wider()`:

```
statclassRe <- statclassTidy %>%
  pivot_wider(names_from = course,
              values_from = grade,
              )

statclassRe
```

```
# A tibble: 3 × 5
  name       r  spss stat1 stat2
  <chr>  <dbl> <dbl> <dbl> <dbl>
1 kim       13    15    14    10
2 momo       6     9    12     5
3 sascha     4     1     7     4
```

`

Tidy-up statclass2

Wo liegt hier noch ein weiteres Problem?

```
statclass2
```

```
   test momo kim sascha  exam
1 stat1   12  13      4 exam1
2 stat1   NA  NA      8 exam2
3 stat2    5  10      5 exam1
4 stat2   NA  NA     NA exam2
5     r    6  13      3 exam1
6     r   NA  NA      9 exam2
7  spss    9   4      7 exam1
8  spss   NA   7     NA exam2
```

→ Namenwerte sind Spaltennamen!

`

Tidy-up statclass2

Und hier die Lösung: Auch hier wandeln wir wieder in das *long*-Format um!

```
statclass2Tidy <- statclass2 %>%
  pivot_longer(momo:sascha,
               names_to = "names",
               values_to = "grade"
               )

statclass2Tidy
```

```
# A tibble: 24 × 4
   test  exam  names  grade
   <chr> <chr> <chr>  <dbl>
 1 stat1 exam1 momo      12
 2 stat1 exam1 kim       13
 3 stat1 exam1 sascha     4
 4 stat1 exam2 momo      NA
 5 stat1 exam2 kim       NA
 6 stat1 exam2 sascha     8
 7 stat2 exam1 momo       5
 8 stat2 exam1 kim       10
 9 stat2 exam1 sascha     5
10 stat2 exam2 momo      NA
   # … with 14 more rows
```

Gibt es evtl. noch mehr Probleme?

`exam` beinhaltet keine Werte, sondern Variablennamen, nämlich `exam1` und `exam2`! Variablen, die die Note in der Prüfung angeben, deren Wert noch in `grade` steht. Deshalb nutzen wir hier jetzt `pivot_wider()`, um die Daten final *tidy* zu machen.

Tidy-up statclass2

```
statclass2Tidy <- statclass2Tidy %>%
  pivot_wider(names_from = exam,
              values_from = grade
              ) %>%
  relocate(names) %>%
  arrange(names,
          test
          )

statclass2Tidy
```

```
# A tibble: 12 × 4
   names  test  exam1 exam2
   <chr>  <chr> <dbl> <dbl>
 1 kim    r        13    NA
 2 kim    spss      4     7
 3 kim    stat1    13    NA
 4 kim    stat2    10    NA
 5 momo   r         6    NA
 6 momo   spss      9    NA
 7 momo   stat1    12    NA
 8 momo   stat2     5    NA
 9 sascha r         3     9
10 sascha spss      7    NA
11 sascha stat1     4     8
12 sascha stat2     5    NA
```

`

Und wieder *wider*

Nur zur Übung könnte man auch dies wiederum in den Ursprungsdatensatz mit `pivot_wider()` verändern:

```
statclass2re <- statclass2Tidy %>%
  pivot_wider(names_from = test,
              values_from = c(exam1,
                              exam2
                              )
              )

statclass2re
```

`

```
# A tibble: 3 x 9
  names  exam1_r exam1_spss exam1_stat1 exam1_stat2 exam2_r exam2_s
  <chr>    <dbl>      <dbl>       <dbl>       <dbl>   <dbl>      <d
1 kim         13          4          13          10      NA
2 momo         6          9          12           5      NA
3 sascha       3          7           4           5       9
# … with 2 more variables: exam2_stat1 <dbl>, exam2_stat2 <dbl>
```

Daten bearbeiten mit dplyr

Teilen eines Datensatzes

Zuerst wiederholen wir noch einmal, wie wir einen Datensatz teilen: Wir filtern die Fälle, die für unsere spätere Analyse relevant sind. Zum Beispiel wollen wir nur über Psychologie-Studierende aus Marburg forschen. Anschließend möchten wir eine neue Variable erstellen, die eine Beschreibung für die Studiendauer inkludiert (als Faktor)

Welche Funktionen müssen wir anwenden?

`

Teilen eines Datensatzes

```
uniPumPsy <- uni %>%
  filter(city == "Marburg" & study == "Psychology") %>%
  mutate(term.group = factor(case_when(term <= 2 ~ "Anfänger:in",
                                       term > 2 & term <= 6 ~ "Regelstudienzeit",
                                       term > 6 ~ "Langzeit"
                                       )
                             )
         )

head(uniPumPsy)
```

```
  ID mot      study    city distance abi term       term.group
1  5   3 Psychology Marburg       43 1.0    8         Langzeit
2  9   6 Psychology Marburg       38 1.3    9         Langzeit
3 27   5 Psychology Marburg       39 1.1    8         Langzeit
4 29   1 Psychology Marburg       44 3.0    2      Anfänger:in
5 61   6 Psychology Marburg       42 4.0    4 Regelstudienzeit
6 67   7 Psychology Marburg       54 2.5    2      Anfänger:in
```

`

Datensätze zusammenführen (Fälle hinzufügen)

Im nächsten Schritt nehmen wir nun an, dass die Datenerfassung von 4 verschiedenen Personen durchgeführt wurde und es somit 4 Teildatensätze gibt, die nun zu einem vollständigen Datensatz verbunden werden sollen. Dazu nutzen wir die Funktion `bind_rows()`. In unserem Beispiel haben alle 4 Teildatensätze genau die gleiche Anzahl an Variablen, die dazu auch noch genau gleich benannt sind! Mit dem Argument `.id` erstellen wir eine Variable names `"origin"`, die die Herkunft des Falles erfasst. Dies ist automatisch nummeriert. Mit `mutate()` machen wir daraus einen Faktor, der eine bessere Beschreibung beinhaltet (*coder1*, *coder2*, *coder3*, *coder4*)

```
uniAll <- uni1 %>%
  bind_rows(list(uni2,
                 uni3,
                 uni4
                 ),
            .id = "origin"
            ) %>%
  mutate(origin = factor(origin,
                         labels = c("coder1",
                                    "coder2",
                                    "coder3",
                                    "coder4"
                                    )
                         )
         )

table(uniAll$origin)
head(uniAll$origin)
```

```
coder1 coder2 coder3 coder4
   250    250    250    250
```

```
[1] coder1 coder1 coder1 coder1 coder1 coder1
Levels: coder1 coder2 coder3 coder4
```

Wir haben hier jetzt also aus vier Teildatensätzen einen gesamten Datensatz erstellt, der alle Fälle der vier Teildatensätze enthält. Wichtig, in diesem Fall waren alle Variablennamen gleich!

Datensätze zusammenführen (Fälle hinzufügen)

Nun probieren wir einmal aus, was passiert, wenn es zum Beispiel in einem Teildatensatz einen Typo gibt. Zuerst erstellen wir dazu einfach zwei neue Datensätze, die jeweils nur 3 Fälle inkludieren, und unterschiedliche Variablen.

```
uniA <- uni[1:3,
            4:5
            ]

City <- c("Giessen",
          "Marburg",
          "Marburg"
          )

distance <- c(21,
              30,
              45
              )

uniB <- data.frame(City,
                   distance
                   )

head(uniA)

head(uniB)
```

```
  city distance
1 Frankfurt       NA
2 Frankfurt       36
3   Marburg       56
```

```
     City distance
1 Giessen       21
2 Marburg       30
3 Marburg       45
```

Wir haben also in beiden Datensätzen die zwei Variablen, die Studienort und die Distanz zum Studienort angeben. im Datensatz `uniB` ist aber die Variable des Studienorts anders geschrieben. Probieren wir `bind_rows()` aus.

Datensätze zusammenführen (Fälle hinzufügen)

```
uniTest <- uniA %>%
  bind_rows(uniB) #<

uniTest
```

```
uniTest <- uniA %>%
  bind_rows(uniB)

uniTest
```

```
       city distance    City
1 Frankfurt       NA    <NA>
2 Frankfurt       36    <NA>
3   Marburg       56    <NA>
4      <NA>       21 Giessen
5      <NA>       30 Marburg
6      <NA>       45 Marburg
```

Wo ist der Fehler?

`

Unterschiedliche Spaltennamen

Wenn die Spalten verschiedene Namen haben, benötigt man das Argument `by`. Hierin werden die Kombinationen der zusammengehörigen Variablen bestimmt.

In unserem Beispiel würden wir also angeben, dass aus Datensatz `uniA` die Spalte `city` gleich der Spalte `City` aus dem Datensatz `uniB` ist. Gleiches gilt für die `distance` Variable.

```
uniTest2 <- uniA %>%
  full_join(uniB,
            by = c("city" = "City",
                   "distance" = "distance"
                   )
            )

head(uniTest2)
```

```
`
```

```
      city distance
1 Frankfurt       NA
2 Frankfurt       36
3   Marburg       56
4   Giessen       21
5   Marburg       30
6   Marburg       45
```

Zwei Datensätze kombinieren

Mögliche Anwendungsfälle:

- mehrere Erhebungen
- Multi-Level-Daten

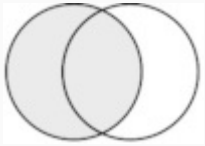
Für das **Mergen** von Datensätzen, kann man je nach Ausgangspunkt `left_join()` bzw. `right_join()` oder auch `full_join()` nutzen. Um die Daten korrekt zu mergen, müssen wir sowohl die Variable `city` als auch `study` nutzen, da sich die Makro-Variablen eben nach Studienort und Studienfach unterscheiden! Dies geben wir im Argument `by` an. Bei `left_join()` geben wir den Datensatz, an dem die Daten hinzugefügt werden sollen, per **Piping** weiter. Bei `right_join()` werden die Daten an den zweiten Datensatz angehängt.

`

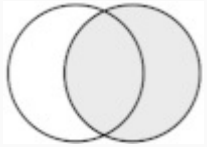
left_join(), right_join() & full_join()

Die Logik der drei Funktionen ist recht gut visuell darzustellen.

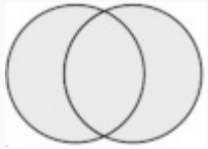
1. **pull-left()**: Daten des linken Datensatzes werden um gemeinsame des rechten Datensatzes ergänzt.



1. **pull-right()**: Daten des rechten Datensatzes werden um gemeinsame des linken Datensatzes ergänzt.



1. **full_join()**: Daten beider Datensätze werden komplett verbunden.



`

Zwei Datensätze kombinieren

Wir möchten nun die Makrodaten aus `uniMacro` jeweils passend auf Studienort und Studienfach in den Mikrodatensatz hinzufügen, um anschließend ein Multi-Level-Modell zu berechnen. Hierzu nutzen wir `left_join()` und geben im Argument `by` an, dass sowohl `city` als auch `study` als **Matching**-Variablen genutzt werden sollen.

```
uniMerged <- uni %>%
  left_join(uniMacro,
            by = c("city",
                   "study"
                   )
            )
uniMerged
```

`

	ID	mot	study	city	distance	abi	term	superv
1	1	8	Political Science	Frankfurt	NA	1.6	3	
2	2	4	Sociology	Frankfurt	36	3.0	5	
3	3	2	Political Science	Marburg	56	2.1	4	
4	4	1	Sociology	Gießen	62	3.3	5	
5	5	3	Psychology	Marburg	43	1.0	8	
6	6	1	Political Science	Marburg	43	1.1	2	
7	7	7	Educational Science	Marburg	39	3.8	3	
8	8	0	Sociology	Marburg	44	2.0	3	
9	9	6	Psychology	Marburg	38	1.3	9	

Zwei Datensätze kombinieren mit full_join()

Alternativ geht dies auch mit `full_join()`:

```
uniMerged2 <- uni %>%
  full_join(uniMacro,
            by = c("city",
                   "study"
                   )
            )
uniMerged2
```

`

	ID	mot	study	city	distance	abi	term	superv
1	1	8	Political Science	Frankfurt	NA	1.6	3	
2	2	4	Sociology	Frankfurt	36	3.0	5	
3	3	2	Political Science	Marburg	56	2.1	4	
4	4	1	Sociology	Gießen	62	3.3	5	
5	5	3	Psychology	Marburg	43	1.0	8	
6	6	1	Political Science	Marburg	43	1.1	2	
7	7	7	Educational Science	Marburg	39	3.8	3	
8	8	0	Sociology	Marburg	44	2.0	3	
9	9	6	Psychology	Marburg	38	1.3	9	

Neue Variablen hinzufügen

Will man nur weitere Variablen in einen Datensatz hinzufügen, kann man auch hierfür `full_join()` nutzen. Wir haben zum Beispiel in einem weiteren Datensatz aus dem Prüfungsverwaltungssystem vor der Anonymisierung der Daten die geleisteten Creditpoints der Befragtena ausgelesen. Diese haben wir im Datensatz `points` getrennt gespeichert und dort ebenfalls eine ID-Variable genutzt, die auf die ID-Variable des Datensatzes `uni` matcht. Wir fügen jetzt die Creditpoints dem Datensatz `uni` mit `full_join()` hinzu. Schauen wir uns zuerst nochmal die zwei Datensätze an:

```
points
```

```
  id ects
1  1   78
2  2   52
3  3  157
4  4   58
5  5   64
6  6   35
7  7   93
```

```
uni
```

```
  ID mot              study      city distance abi term
1  1   8  Political Science Frankfurt       NA 1.6    3
2  2   4          Sociology Frankfurt       36 3.0    5
3  3   2  Political Science   Marburg       56 2.1    4
4  4   1          Sociology    Gießen       62 3.3    5
5  5   3        Psychology   Marburg       43 1.0    8
6  6   1  Political Science   Marburg       43 1.1    2
7  7   7 Educational Science   Marburg       39 3.8    3
```

Wir haben zwar in beiden Variablen eine ID-Variable, allerdings ist die Spalte unterschiedlich benannt.

`

Neue Variablen hinzufügen

Wir können jetzt - wie zuvor oben - wieder im `by`-Argument dies angeben. Diesmal wollen wir einfach schnell vorher den Spaltennamen in einem der Datensätze anpassen. Dazu nutzen wir einfach `rename()`. Die Logik in der Funktion ist `neuer Name = alter Name`. Dann sind die Spaltennamen gleich und wir können die Datensätze mergen.

```
points <- points %>%
  rename(ID = id)

uni <- uni %>%
  full_join(points,
            by = "ID"
            )
uni
```

`

	ID	mot	study	city	distance	abi	term	ects
1	1	8	Political Science	Frankfurt	NA	1.6	3	78
2	2	4	Sociology	Frankfurt	36	3.0	5	52
3	3	2	Political Science	Marburg	56	2.1	4	157
4	4	1	Sociology	Gießen	62	3.3	5	58
5	5	3	Psychology	Marburg	43	1.0	8	64
6	6	1	Political Science	Marburg	43	1.1	2	35
7	7	7	Educational Science	Marburg	39	3.8	3	93
8	8	0	Sociology	Marburg	44	2.0	3	137
9	9	6	Psychology	Marburg	38	1.3	9	150

Das war's! Nun machen wir ein kleines Quiz!